In [1]:
```python
import pandas as pd
import numpy as np
```

In [2]:
```python
raw_df = pd.read_csv("loan_data.csv")
raw_df.shape
```

```
(9578, 14)
```

In [3]:
```python
#Getting a Statstical summary of data
raw_df.describe()
```

	credit.policy	int.rate	installment	log.annual.inc	dti	fico	days.with.cr.line	revol.bal	revol.util
count	9578.000000	9578.000000	9578.000000	9578.000000	9578.000000	9578.000000	9578.000000	9.578000e+03	9578.000000
mean	0.804970	0.122640	319.089413	10.932117	12.606679	710.846314	4560.767197	1.691396e+04	46.799236
std	0.396245	0.026847	207.071301	0.614813	6.883970	37.970537	2496.930377	3.375619e+04	29.014417
min	0.000000	0.060000	15.670000	7.547502	0.000000	612.000000	178.958333	0.000000e+00	0.000000
25%	1.000000	0.103900	163.770000	10.558414	7.212500	682.000000	2820.000000	3.187000e+03	22.600000
50%	1.000000	0.122100	268.950000	10.928884	12.665000	707.000000	4139.958333	8.596000e+03	46.300000
75%	1.000000	0.140700	432.762500	11.291293	17.950000	737.000000	5730.000000	1.824950e+04	70.900000
max	1.000000	0.216400	940.140000	14.528354	29.960000	827.000000	17639.958330	1.207359e+06	119.000000

In [4]:
```python
# Getting the datatype and checking for null values if any
raw_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9578 entries, 0 to 9577
Data columns (total 14 columns):
credit.policy        9578 non-null int64
purpose              9578 non-null object
int.rate             9578 non-null float64
installment          9578 non-null float64
log.annual.inc       9578 non-null float64
dti                  9578 non-null float64
fico                 9578 non-null int64
days.with.cr.line    9578 non-null float64
revol.bal            9578 non-null int64
revol.util           9578 non-null float64
inq.last.6mths       9578 non-null int64
delinq.2yrs          9578 non-null int64
pub.rec              9578 non-null int64
not.fully.paid       9578 non-null int64
dtypes: float64(6), int64(7), object(1)
memory usage: 1.0+ MB
```

In [5]:
```python
raw_df['purpose'].value_counts()
```

```
debt_consolidation    3957
all_other             2331
credit_card           1262
home_improvement       629
small_business         619
major_purchase         437
educational            343
Name: purpose, dtype: int64
```

In [6]:
```
# Encoding the categorical values to numerical
purpose_encoded = pd.get_dummies(raw_df['purpose'])
purpose_encoded.columns = ['purpose_all_other','purpose_credit_card','purpose_debt_consolidation',
                           'purpose_educational','purpose_home_improvement','purpose_major_purchas
e',
                          'purpose_small_business']
purpose_encoded.head()
```

	purpose_all_other	purpose_credit_card	purpose_debt_consolidation	purpose_educational	purpose_home_improvement	pu
0	0	0	1	0	0	0
1	0	1	0	0	0	0
2	0	0	1	0	0	0
3	0	0	1	0	0	0
4	0	1	0	0	0	0

In [7]:
```
# adding the encoded columns to the df and droping the purpose column
raw_df = raw_df.join(purpose_encoded)
raw_df.drop(['purpose'],axis=1,inplace=True)
raw_df.head()
```

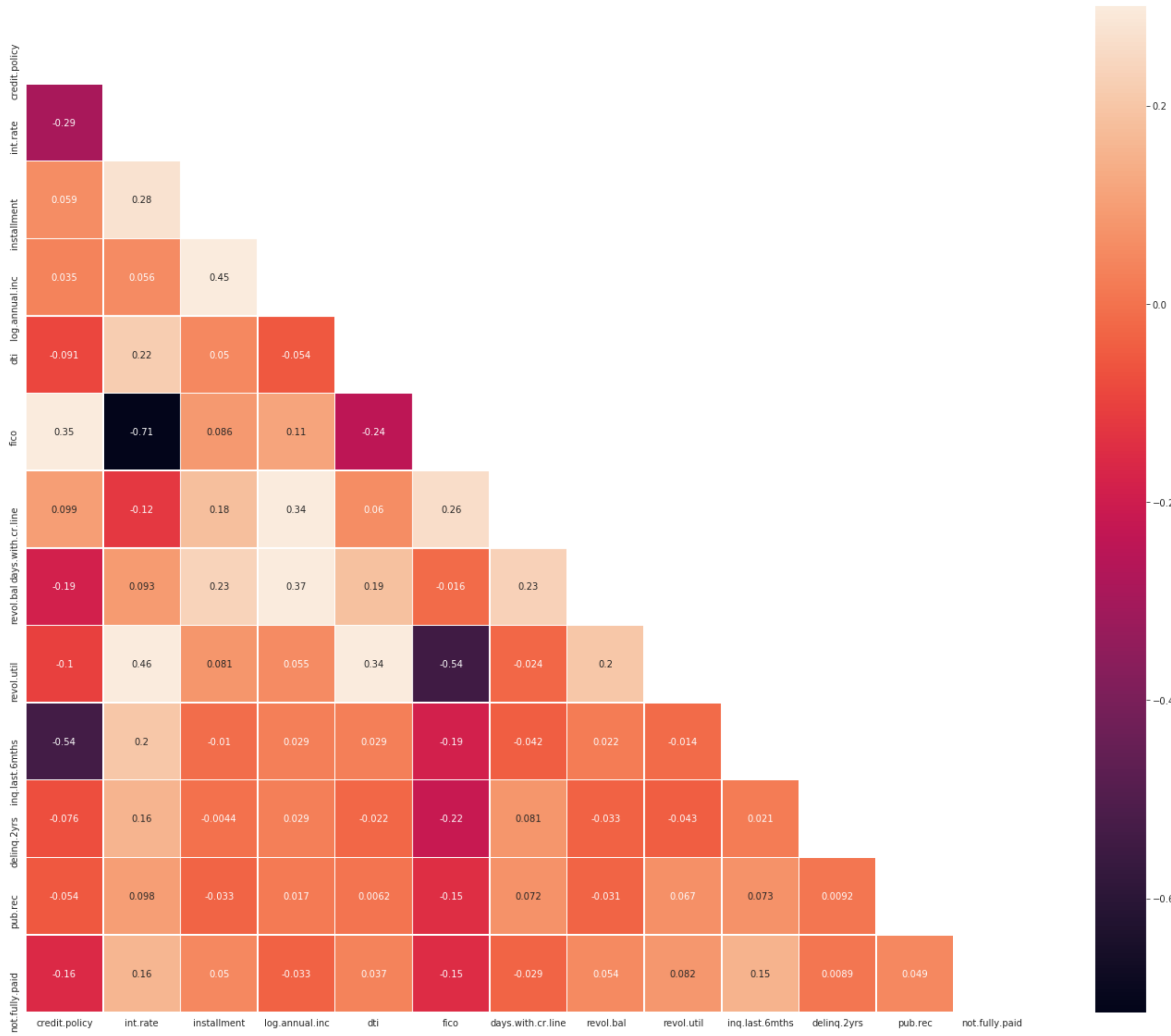
	credit.policy	int.rate	installment	log.annual.inc	dti	fico	days.with.cr.line	revol.bal	revol.util	inq.last.6mths	delinq.2yı
0	1	0.1189	829.10	11.350407	19.48	737	5639.958333	28854	52.1	0	0
1	1	0.1071	228.22	11.082143	14.29	707	2760.000000	33623	76.7	0	0
2	1	0.1357	366.86	10.373491	11.63	682	4710.000000	3511	25.6	1	0
3	1	0.1008	162.34	11.350407	8.10	712	2699.958333	33667	73.2	1	0
4	1	0.1426	102.92	11.299732	14.97	667	4066.000000	4740	39.5	0	1

In [8]:
```
raw_df['credit.policy'].value_counts()
```

```
1    7710
0    1868
Name: credit.policy, dtype: int64
```

In [9]:
```
c = ['credit.policy','int.rate','installment','log.annual.inc','dti','fico','days.with.cr.line','r
evol.bal','revol.util','inq.last.6mths','delinq.2yrs','pub.rec','not.fully.paid']
temp_df = raw_df.loc[:,c]
corr = temp_df.corr()
```

In [10]:
```
import seaborn as sns
import matplotlib.pyplot as plt
mask = np.zeros_like(corr)
mask[np.triu_indices_from(mask)] = True
with sns.axes_style("white"):
    f, ax = plt.subplots(figsize=(25,20))
    ax = sns.heatmap(corr, mask=mask, vmax=.3, square=True,annot=True,linewidth=.5)
```



In [11]:
```
# removing features with (+-)0.7 correlation -> fico and int.rate are highly correlated so removin
g fico
raw_df = raw_df.drop(['fico'],axis=1)
raw_df.head()
```

	credit.policy	int.rate	installment	log.annual.inc	dti	days.with.cr.line	revol.bal	revol.util	inq.last.6mths	delinq.2yrs	pu
0	1	0.1189	829.10	11.350407	19.48	5639.958333	28854	52.1	0	0	0
1	1	0.1071	228.22	11.082143	14.29	2760.000000	33623	76.7	0	0	0
2	1	0.1357	366.86	10.373491	11.63	4710.000000	3511	25.6	1	0	0
3	1	0.1008	162.34	11.350407	8.10	2699.958333	33667	73.2	1	0	0
4	1	0.1426	102.92	11.299732	14.97	4066.000000	4740	39.5	0	1	0

In [12]:
```
cols = raw_df.shape[1]-1
x = raw_df.drop(['credit.policy'],axis=1).values
y = raw_df['credit.policy'].values
x.shape,y.shape,raw_df.shape,cols
```

```
((9578, 18), (9578,), (9578, 19), 18)
```

In [13]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_t = sc.fit_transform(x)
x_t
```

```
array([[-0.13931753,  2.46309947,  0.68038804, ..., -0.2651173 ,
        -0.21864717, -0.26285458],
       [-0.57886837, -0.43885443,  0.2440308 , ..., -0.2651173 ,
        -0.21864717, -0.26285458],
       [ 0.48648368,  0.23070836, -0.90865897, ..., -0.2651173 ,
        -0.21864717, -0.26285458],
       ...,
       [-0.57886837, -1.06867038, -0.54569448, ..., -0.2651173 ,
        -0.21864717, -0.26285458],
       [ 1.39166043,  0.1569135 , -0.18272998, ...,  3.77191529,
        -0.21864717, -0.26285458],
       [ 0.61685894,  2.58060136,  0.54059439, ..., -0.2651173 ,
        -0.21864717, -0.26285458]])
```

In [14]:
```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import BatchNormalization
from keras.optimizers import SGD, Adam
from keras.layers import Dropout
```

```
Using TensorFlow backend.
```

In [15]:
```
model = Sequential()
model.add(Dense(20, input_dim=cols, activation='relu', kernel_initializer='he_uniform'))
model.add(BatchNormalization())
model.add(Dense(30,kernel_initializer='he_uniform', activation='relu'))
model.add(Dense(20,kernel_initializer='he_uniform', activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

In [16]:
```
epoch=50
lr = 0.001
decay_rate = lr / epoch
momentum = 0.9
sgd = SGD(lr=lr, momentum=momentum, decay=decay_rate, nesterov=False)
model.compile(loss='binary_crossentropy', optimizer=sgd, metrics=['accuracy'])
```

In [17]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x_t, y, test_size=0.30, random_state=1)
x_train.shape, x_test.shape
```

```
((6704, 18), (2874, 18))
```

In [18]:

```
batch = int(x_train.shape[0]/50)
nn_model = model.fit(x_train, y_train,validation_data=(x_test, y_test), epochs=epoch,batch_size =
batch)
```

```
Train on 6704 samples, validate on 2874 samples
Epoch 1/50
6704/6704 [==============================] - 0s 57us/step - loss: 0.7355 - accuracy: 0.5837 - val_loss: 0.5659 - val_accuracy: 0.7683
Epoch 2/50
6704/6704 [==============================] - 0s 16us/step - loss: 0.5299 - accuracy: 0.7874 - val_loss: 0.4982 - val_accuracy: 0.8072
Epoch 3/50
6704/6704 [==============================] - 0s 16us/step - loss: 0.4807 - accuracy: 0.8112 - val_loss: 0.4592 - val_accuracy: 0.8267
Epoch 4/50
6704/6704 [==============================] - 0s 16us/step - loss: 0.4492 - accuracy: 0.8234 - val_loss: 0.4309 - val_accuracy: 0.8340
Epoch 5/50
6704/6704 [==============================] - 0s 18us/step - loss: 0.4265 - accuracy: 0.8374 - val_loss: 0.4087 - val_accuracy: 0.8473
Epoch 6/50
6704/6704 [==============================] - 0s 19us/step - loss: 0.4069 - accuracy: 0.8408 - val_loss: 0.3908 - val_accuracy: 0.8546
Epoch 7/50
6704/6704 [==============================] - 0s 18us/step - loss: 0.3934 - accuracy: 0.8511 - val_loss: 0.3773 - val_accuracy: 0.8580
Epoch 8/50
6704/6704 [==============================] - 0s 17us/step - loss: 0.3822 - accuracy: 0.8550 - val_loss: 0.3649 - val_accuracy: 0.8615
Epoch 9/50
6704/6704 [==============================] - 0s 17us/step - loss: 0.3706 - accuracy: 0.8556 - val_loss: 0.3557 - val_accuracy: 0.8612
Epoch 10/50
6704/6704 [==============================] - 0s 16us/step - loss: 0.3631 - accuracy: 0.8572 - val_loss: 0.3460 - val_accuracy: 0.8667
Epoch 11/50
6704/6704 [==============================] - 0s 16us/step - loss: 0.3525 - accuracy: 0.8646 - val_loss: 0.3371 - val_accuracy: 0.8695
Epoch 12/50
6704/6704 [==============================] - 0s 16us/step - loss: 0.3462 - accuracy: 0.8661 - val_loss: 0.3300 - val_accuracy: 0.8737
Epoch 13/50
6704/6704 [==============================] - 0s 18us/step - loss: 0.3400 - accuracy: 0.8707 - val_loss: 0.3230 - val_accuracy: 0.8772
Epoch 14/50
6704/6704 [==============================] - 0s 16us/step - loss: 0.3325 - accuracy: 0.8705 - val_loss: 0.3171 - val_accuracy: 0.8807
Epoch 15/50
6704/6704 [==============================] - 0s 17us/step - loss: 0.3270 - accuracy: 0.8740 - val_loss: 0.3117 - val_accuracy: 0.8827
Epoch 16/50
6704/6704 [==============================] - 0s 19us/step - loss: 0.3249 - accuracy: 0.8719 - val_loss: 0.3067 - val_accuracy: 0.8866
Epoch 17/50
6704/6704 [==============================] - 0s 16us/step - loss: 0.3173 - accuracy: 0.8787 - val_loss: 0.3028 - val_accuracy: 0.8869
Epoch 18/50
6704/6704 [==============================] - 0s 17us/step - loss: 0.3159 - accuracy: 0.8747 - val_loss: 0.2994 - val_accuracy: 0.8883
Epoch 19/50
6704/6704 [==============================] - 0s 16us/step - loss: 0.3121 - accuracy: 0.8802 - val_loss: 0.2963 - val_accuracy: 0.8887
Epoch 20/50
6704/6704 [==============================] - 0s 16us/step - loss: 0.3102 - accuracy: 0.8798 - val_loss: 0.2942 - val_accuracy: 0.8897
Epoch 21/50
6704/6704 [==============================] - 0s 16us/step - loss: 0.3050 - accuracy: 0.8828 - val_loss: 0.2920 - val_accuracy: 0.8897
Epoch 22/50
6704/6704 [==============================] - 0s 18us/step - loss: 0.3022 - accuracy: 0.8814 - val_loss: 0.2897 - val_accuracy: 0.8918
Epoch 23/50
6704/6704 [==============================] - 0s 17us/step - loss: 0.2995 - accuracy: 0.8853 - val_loss: 0.2875 - val_accuracy: 0.8928
Epoch 24/50
6704/6704 [==============================] - 0s 24us/step - loss: 0.3001 - accuracy: 0.8835 - val_loss: 0.2860 - val_accuracy: 0.8921
Epoch 25/50
6704/6704 [==============================] - 0s 16us/step - loss: 0.2953 - accuracy: 0.8841 - val_loss: 0.2855 - val_accuracy: 0.8904
Epoch 26/50
6704/6704 [==============================] - 0s 16us/step - loss: 0.2950 - accuracy: 0.8857 - val_loss: 0.2834 - val_accuracy: 0.8914
Epoch 27/50
6704/6704 [==============================] - 0s 17us/step - loss: 0.2958 - accuracy: 0.8817 - val_loss: 0.2815 - val_accuracy: 0.8932
Epoch 28/50
6704/6704 [==============================] - 0s 17us/step - loss: 0.2920 - accuracy: 0.8862 - val_loss: 0.2801 - val_accuracy: 0.8918
Epoch 29/50
6704/6704 [==============================] - 0s 16us/step - loss: 0.2933 - accuracy: 0.8860 - val_loss: 0.2790 - val_accuracy: 0.8928
Epoch 30/50
6704/6704 [==============================] - 0s 18us/step - loss: 0.2888 - accuracy: 0.8856 - val_loss: 0.2776 - val_accuracy: 0.8932
Epoch 31/50
6704/6704 [==============================] - 0s 17us/step - loss: 0.2892 - accuracy: 0.8851 - val_loss: 0.2766 - val_accuracy: 0.8918
Epoch 32/50
6704/6704 [==============================] - 0s 16us/step - loss: 0.2870 - accuracy: 0.8889 - val_loss: 0.2764 - val_accuracy: 0.8935
Epoch 33/50
6704/6704 [==============================] - 0s 16us/step - loss: 0.2863 - accuracy: 0.8859 - val_loss: 0.2763 - val_accuracy: 0.8928
Epoch 34/50
6704/6704 [==============================] - 0s 16us/step - loss: 0.2865 - accuracy: 0.8853 - val_loss: 0.2750 - val_accuracy: 0.8935
Epoch 35/50
6704/6704 [==============================] - 0s 16us/step - loss: 0.2859 - accuracy: 0.8865 - val_loss: 0.2741 - val_accuracy: 0.8935
Epoch 36/50
6704/6704 [==============================] - 0s 16us/step - loss: 0.2810 - accuracy: 0.8892 - val_loss: 0.2731 - val_accuracy: 0.8932
Epoch 37/50
6704/6704 [==============================] - 0s 16us/step - loss: 0.2826 - accuracy: 0.8881 - val_loss: 0.2721 - val_accuracy: 0.8928
Epoch 38/50
```

```
6704/6704 [==============================] - 0s 16us/step - loss: 0.2836 - accuracy: 0.8881 - val_loss: 0.2710 - val_accuracy: 0.8939
Epoch 39/50
6704/6704 [==============================] - 0s 18us/step - loss: 0.2789 - accuracy: 0.8893 - val_loss: 0.2699 - val_accuracy: 0.8949
Epoch 40/50
6704/6704 [==============================] - 0s 16us/step - loss: 0.2764 - accuracy: 0.8883 - val_loss: 0.2689 - val_accuracy: 0.8960
Epoch 41/50
6704/6704 [==============================] - 0s 16us/step - loss: 0.2769 - accuracy: 0.8892 - val_loss: 0.2683 - val_accuracy: 0.8956
Epoch 42/50
6704/6704 [==============================] - 0s 16us/step - loss: 0.2741 - accuracy: 0.8925 - val_loss: 0.2674 - val_accuracy: 0.8967
Epoch 43/50
6704/6704 [==============================] - 0s 16us/step - loss: 0.2751 - accuracy: 0.8920 - val_loss: 0.2669 - val_accuracy: 0.8949
Epoch 44/50
6704/6704 [==============================] - 0s 16us/step - loss: 0.2720 - accuracy: 0.8939 - val_loss: 0.2664 - val_accuracy: 0.8949
Epoch 45/50
6704/6704 [==============================] - 0s 23us/step - loss: 0.2736 - accuracy: 0.8925 - val_loss: 0.2659 - val_accuracy: 0.8953
Epoch 46/50
6704/6704 [==============================] - 0s 35us/step - loss: 0.2716 - accuracy: 0.8929 - val_loss: 0.2652 - val_accuracy: 0.8974
Epoch 47/50
6704/6704 [==============================] - 0s 32us/step - loss: 0.2723 - accuracy: 0.8917 - val_loss: 0.2645 - val_accuracy: 0.8960
Epoch 48/50
6704/6704 [==============================] - 0s 27us/step - loss: 0.2670 - accuracy: 0.8939 - val_loss: 0.2642 - val_accuracy: 0.8977
Epoch 49/50
6704/6704 [==============================] - 0s 31us/step - loss: 0.2693 - accuracy: 0.8939 - val_loss: 0.2634 - val_accuracy: 0.8984
Epoch 50/50
6704/6704 [==============================] - 0s 32us/step - loss: 0.2677 - accuracy: 0.8941 - val_loss: 0.2631 - val_accuracy: 0.8987
```

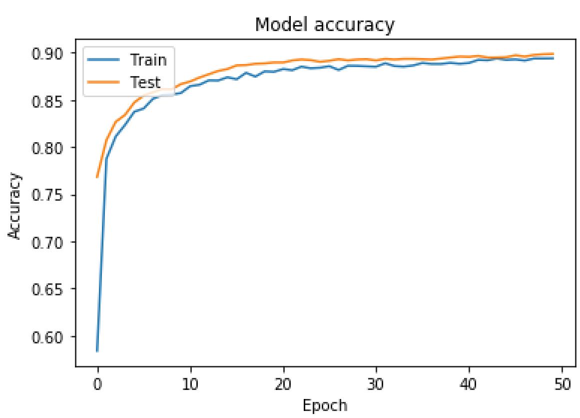
In [19]:

```python
_, train_acc = model.evaluate(x_train, y_train, verbose=0)
_, test_acc = model.evaluate(x_test, y_test, verbose=0)
print('Train: %.3f, Test: %.3f' % (train_acc, test_acc))
```

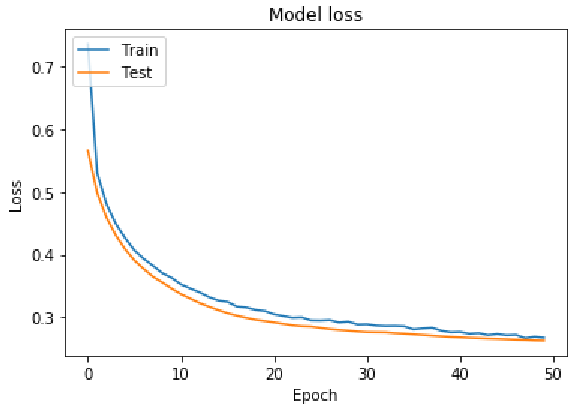
```
Train: 0.898, Test: 0.899
```

In [20]:

```python
import matplotlib.pyplot as plt
plt.plot(nn_model.history['accuracy'])
plt.plot(nn_model.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```



In [21]:
```
plt.plot(nn_model.history['loss'])
plt.plot(nn_model.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```



Hyper parameter tunning

In [23]:
```
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import GridSearchCV
```

In [24]:
```
s = x_train.shape[0]
init_mode = ['glorot_normal', 'glorot_uniform', 'he_uniform']
batch_size = [ int(s/20), int(s/50)]
epochs = [75,100]
optimizer = ['SGD', 'RMSprop', 'Adam']
learn_rate = [0.001, 0.01]
momentum = [0.0, 0.5, 0.9]
activation = ['softplus', 'softmax', 'relu']
dropout_rate = [0.2, 0.4, 0.6, 0.8, 0.9]
from keras.constraints import maxnorm
def create_model(init_mode='glorot_normal',activation='relu',dropout_rate=0.0, learn_rate=0.001,
                neuron=15,optimizer='Adam'):
    model = Sequential()
    model.add(Dense(neuron, input_dim=cols, activation=activation, kernel_initializer=init_mode))
    model.add(BatchNormalization())
    model.add(Dense(neuron, activation=activation))
    model.add(Dropout(dropout_rate))
    model.add(Dense(neuron, activation=activation))
    model.add(Dense(1, activation='sigmoid'))
    optimizer = Adam(learning_rate=learn_rate)
    model.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['accuracy'])
    return model

model_CV = KerasClassifier(build_fn=create_model, verbose=0,epochs=75, batch_size=134)
param_grid = dict(init_mode=init_mode,batch_size=batch_size,epochs=epochs,activation=activation)
grid_search = GridSearchCV(estimator=model_CV, param_grid=param_grid,cv=3,
                           scoring='accuracy', verbose=10, n_jobs=2)
result = grid_search.fit(x_train, y_train)
```

```
Fitting 3 folds for each of 36 candidates, totalling 108 fits


[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done   1 tasks      | elapsed:   15.6s
[Parallel(n_jobs=2)]: Done   4 tasks      | elapsed:   20.8s
[Parallel(n_jobs=2)]: Done   9 tasks      | elapsed:   36.7s
[Parallel(n_jobs=2)]: Done  14 tasks      | elapsed:   51.5s
[Parallel(n_jobs=2)]: Done  21 tasks      | elapsed:  1.4min
[Parallel(n_jobs=2)]: Done  28 tasks      | elapsed:  1.9min
[Parallel(n_jobs=2)]: Done  37 tasks      | elapsed:  2.8min
[Parallel(n_jobs=2)]: Done  46 tasks      | elapsed:  3.2min
[Parallel(n_jobs=2)]: Done  57 tasks      | elapsed:  4.1min
[Parallel(n_jobs=2)]: Done  68 tasks      | elapsed:  5.2min
[Parallel(n_jobs=2)]: Done  81 tasks      | elapsed:  6.1min
[Parallel(n_jobs=2)]: Done  94 tasks      | elapsed:  6.9min
[Parallel(n_jobs=2)]: Done 108 out of 108 | elapsed:  8.3min finished
```

In [25]:

```
print(f'Best Accuracy for {result.best_score_} using {result.best_params_}\n')
means = result.cv_results_['mean_test_score']
stds = result.cv_results_['std_test_score']
params = result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print(f' mean={mean:.4}, std={stdev:.4} using {param}')
```

```
Best Accuracy for 0.9197500896256553 using {'activation': 'softmax', 'batch_size': 134, 'epochs': 75, 'init_mode': 'glorot_uniform'}

 mean=0.9056, std=0.003544 using {'activation': 'softplus', 'batch_size': 335, 'epochs': 75, 'init_mode': 'glorot_normal'}
 mean=0.9007, std=0.0003837 using {'activation': 'softplus', 'batch_size': 335, 'epochs': 75, 'init_mode': 'glorot_uniform'}
 mean=0.9021, std=0.0005623 using {'activation': 'softplus', 'batch_size': 335, 'epochs': 75, 'init_mode': 'he_uniform'}
 mean=0.9017, std=0.002442 using {'activation': 'softplus', 'batch_size': 335, 'epochs': 100, 'init_mode': 'glorot_normal'}
 mean=0.9021, std=0.0009007 using {'activation': 'softplus', 'batch_size': 335, 'epochs': 100, 'init_mode': 'glorot_uniform'}
 mean=0.9023, std=0.003717 using {'activation': 'softplus', 'batch_size': 335, 'epochs': 100, 'init_mode': 'he_uniform'}
 mean=0.9083, std=0.00417 using {'activation': 'softplus', 'batch_size': 134, 'epochs': 75, 'init_mode': 'glorot_normal'}
 mean=0.9081, std=0.004013 using {'activation': 'softplus', 'batch_size': 134, 'epochs': 75, 'init_mode': 'glorot_uniform'}
 mean=0.9083, std=0.002512 using {'activation': 'softplus', 'batch_size': 134, 'epochs': 75, 'init_mode': 'he_uniform'}
 mean=0.9112, std=0.003675 using {'activation': 'softplus', 'batch_size': 134, 'epochs': 100, 'init_mode': 'glorot_normal'}
 mean=0.9093, std=0.001372 using {'activation': 'softplus', 'batch_size': 134, 'epochs': 100, 'init_mode': 'glorot_uniform'}
 mean=0.9047, std=0.001678 using {'activation': 'softplus', 'batch_size': 134, 'epochs': 100, 'init_mode': 'he_uniform'}
 mean=0.8971, std=0.001601 using {'activation': 'softmax', 'batch_size': 335, 'epochs': 75, 'init_mode': 'glorot_normal'}
 mean=0.9038, std=0.001096 using {'activation': 'softmax', 'batch_size': 335, 'epochs': 75, 'init_mode': 'glorot_uniform'}
 mean=0.9013, std=0.004018 using {'activation': 'softmax', 'batch_size': 335, 'epochs': 75, 'init_mode': 'he_uniform'}
 mean=0.9023, std=0.004975 using {'activation': 'softmax', 'batch_size': 335, 'epochs': 100, 'init_mode': 'glorot_normal'}
 mean=0.9059, std=0.00174 using {'activation': 'softmax', 'batch_size': 335, 'epochs': 100, 'init_mode': 'glorot_uniform'}
 mean=0.9013, std=0.008754 using {'activation': 'softmax', 'batch_size': 335, 'epochs': 100, 'init_mode': 'he_uniform'}
 mean=0.9065, std=0.005727 using {'activation': 'softmax', 'batch_size': 134, 'epochs': 75, 'init_mode': 'glorot_normal'}
 mean=0.9198, std=0.003299 using {'activation': 'softmax', 'batch_size': 134, 'epochs': 75, 'init_mode': 'glorot_uniform'}
 mean=0.9039, std=0.004102 using {'activation': 'softmax', 'batch_size': 134, 'epochs': 75, 'init_mode': 'he_uniform'}
 mean=0.9156, std=0.00697 using {'activation': 'softmax', 'batch_size': 134, 'epochs': 100, 'init_mode': 'glorot_normal'}
 mean=0.9081, std=0.009411 using {'activation': 'softmax', 'batch_size': 134, 'epochs': 100, 'init_mode': 'glorot_uniform'}
 mean=0.9139, std=0.0009363 using {'activation': 'softmax', 'batch_size': 134, 'epochs': 100, 'init_mode': 'he_uniform'}
 mean=0.9026, std=0.002359 using {'activation': 'relu', 'batch_size': 335, 'epochs': 75, 'init_mode': 'glorot_normal'}
 mean=0.9068, std=0.002453 using {'activation': 'relu', 'batch_size': 335, 'epochs': 75, 'init_mode': 'glorot_uniform'}
 mean=0.8987, std=0.002437 using {'activation': 'relu', 'batch_size': 335, 'epochs': 75, 'init_mode': 'he_uniform'}
 mean=0.9026, std=0.001045 using {'activation': 'relu', 'batch_size': 335, 'epochs': 100, 'init_mode': 'glorot_normal'}
 mean=0.9021, std=0.003125 using {'activation': 'relu', 'batch_size': 335, 'epochs': 100, 'init_mode': 'glorot_uniform'}
 mean=0.902, std=0.008861 using {'activation': 'relu', 'batch_size': 335, 'epochs': 100, 'init_mode': 'he_uniform'}
 mean=0.8998, std=0.0055 using {'activation': 'relu', 'batch_size': 134, 'epochs': 75, 'init_mode': 'glorot_normal'}
 mean=0.9069, std=0.001671 using {'activation': 'relu', 'batch_size': 134, 'epochs': 75, 'init_mode': 'glorot_uniform'}
 mean=0.9017, std=0.006831 using {'activation': 'relu', 'batch_size': 134, 'epochs': 75, 'init_mode': 'he_uniform'}
 mean=0.9062, std=0.003713 using {'activation': 'relu', 'batch_size': 134, 'epochs': 100, 'init_mode': 'glorot_normal'}
 mean=0.9072, std=0.00681 using {'activation': 'relu', 'batch_size': 134, 'epochs': 100, 'init_mode': 'glorot_uniform'}
 mean=0.9038, std=0.004562 using {'activation': 'relu', 'batch_size': 134, 'epochs': 100, 'init_mode': 'he_uniform'}
```

In [28]:
```
optimizer = ['SGD', 'RMSprop', 'Adam']
learn_rate = [0.001, 0.01]
momentum = [0.0, 0.5, 0.9]
dropout_rate = [0.2, 0.4, 0.6, 0.8, 0.9]
# neurons = [10,15,20,25,30]
from keras.constraints import maxnorm
def create_model(init_mode='he_uniform',activation='relu',dropout_rate=0.0,
                neuron=15,optimizer='Adam'):
    model = Sequential()
    model.add(Dense(neuron, input_dim=cols, activation=activation, kernel_initializer=init_mode))
    model.add(BatchNormalization())
    model.add(Dense(neuron, activation=activation))
    model.add(Dropout(dropout_rate))
    model.add(Dense(neuron, activation=activation))
    model.add(Dense(1, activation='sigmoid'))
    #optimizer = Adam(learning_rate=learn_rate)
    model.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['accuracy'])
    return model


model_CV = KerasClassifier(build_fn=create_model, verbose=0,epochs=100, batch_size=30)
param_grid = dict(optimizer=optimizer,dropout_rate=dropout_rate)
grid_search = GridSearchCV(estimator=model_CV, param_grid=param_grid,cv=3,
                           scoring='accuracy', verbose=10,n_jobs=2)
result = grid_search.fit(x_train, y_train)
```

```
Fitting 3 folds for each of 15 candidates, totalling 45 fits


[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done   1 tasks      | elapsed:   53.6s
[Parallel(n_jobs=2)]: Done   4 tasks      | elapsed:  1.5min
[Parallel(n_jobs=2)]: Done   9 tasks      | elapsed:  3.4min
[Parallel(n_jobs=2)]: Done  14 tasks      | elapsed:  4.5min
[Parallel(n_jobs=2)]: Done  21 tasks      | elapsed:  6.8min
[Parallel(n_jobs=2)]: Done  28 tasks      | elapsed:  8.7min
[Parallel(n_jobs=2)]: Done  37 tasks      | elapsed: 11.5min
[Parallel(n_jobs=2)]: Done  45 out of  45 | elapsed: 13.8min finished
```

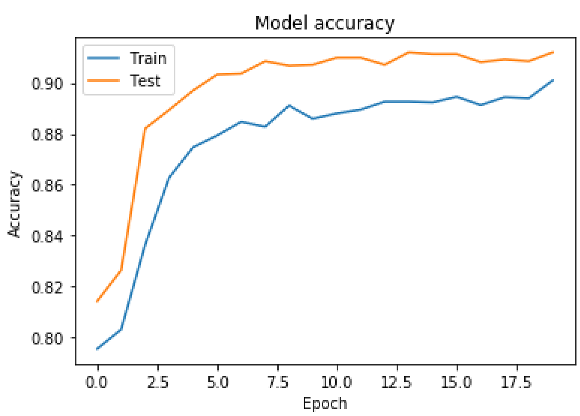
In [29]:
```
print(f'Best Accuracy for {result.best_score_} using {result.best_params_}\n')
means = result.cv_results_['mean_test_score']
stds = result.cv_results_['std_test_score']
params = result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print(f' mean={mean:.4}, std={stdev:.4} using {param}')
```

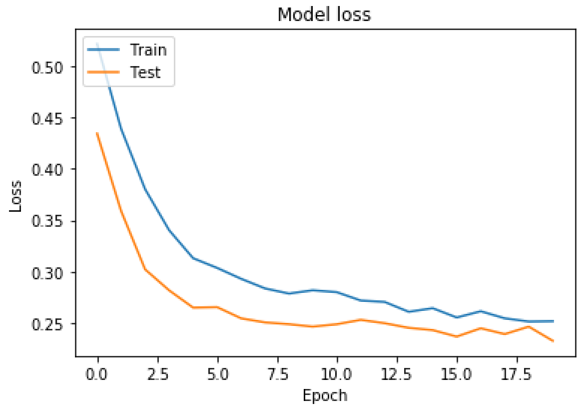
```
Best Accuracy for 0.909756078021386 using {'dropout_rate': 0.2, 'optimizer': 'Adam'}

 mean=0.8999, std=0.003507 using {'dropout_rate': 0.2, 'optimizer': 'SGD'}
 mean=0.9078, std=0.002881 using {'dropout_rate': 0.2, 'optimizer': 'RMSprop'}
 mean=0.9098, std=0.005236 using {'dropout_rate': 0.2, 'optimizer': 'Adam'}
 mean=0.8987, std=0.003492 using {'dropout_rate': 0.4, 'optimizer': 'SGD'}
 mean=0.9056, std=0.004212 using {'dropout_rate': 0.4, 'optimizer': 'RMSprop'}
 mean=0.905, std=0.004727 using {'dropout_rate': 0.4, 'optimizer': 'Adam'}
 mean=0.8743, std=0.01885 using {'dropout_rate': 0.6, 'optimizer': 'SGD'}
 mean=0.8993, std=0.01252 using {'dropout_rate': 0.6, 'optimizer': 'RMSprop'}
 mean=0.8966, std=0.001888 using {'dropout_rate': 0.6, 'optimizer': 'Adam'}
 mean=0.8477, std=0.02888 using {'dropout_rate': 0.8, 'optimizer': 'SGD'}
 mean=0.8799, std=0.01789 using {'dropout_rate': 0.8, 'optimizer': 'RMSprop'}
 mean=0.8599, std=0.007645 using {'dropout_rate': 0.8, 'optimizer': 'Adam'}
 mean=0.8164, std=0.01314 using {'dropout_rate': 0.9, 'optimizer': 'SGD'}
 mean=0.8167, std=0.01436 using {'dropout_rate': 0.9, 'optimizer': 'RMSprop'}
 mean=0.8376, std=0.007484 using {'dropout_rate': 0.9, 'optimizer': 'Adam'}
```

Final Model Neural Network

In [30]:
```
dr = 0.2
model = Sequential()
model.add(Dense(20, input_dim=cols, activation='relu', kernel_initializer='he_uniform'))
model.add(BatchNormalization())
model.add(Dense(30,kernel_initializer='he_uniform', activation='relu'))
model.add(Dropout(dr))
model.add(Dense(20,kernel_initializer='he_uniform', activation='relu'))
model.add(Dropout(dr))
model.add(Dense(20,kernel_initializer='he_uniform', activation='relu'))
model.add(Dropout(dr))
model.add(Dense(20,kernel_initializer='he_uniform', activation='relu'))
model.add(Dense(1, activation='sigmoid'))
epoch=20
lr = 0.001
decay_rate = lr / epoch
momentum = 0.9
optimizer = Adam(learning_rate=lr)
model.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['accuracy'])
batch = 30 #int(x_train.shape[0]/50)
nn_model = model.fit(x_train, y_train,validation_data=(x_test, y_test), epochs=epoch,batch_size =
batch)
_, train_acc = model.evaluate(x_train, y_train, verbose=0)
_, test_acc = model.evaluate(x_test, y_test, verbose=0)
print('Train: %.3f, Test: %.3f' % (train_acc, test_acc))
plt.plot(nn_model.history['accuracy'])
plt.plot(nn_model.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
plt.plot(nn_model.history['loss'])
plt.plot(nn_model.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```

```
Train on 6704 samples, validate on 2874 samples
Epoch 1/20
6704/6704 [==============================] - 1s 189us/step - loss: 0.5216 - accuracy: 0.7955 - val_loss: 0.4343 - val_accuracy: 0.8142
Epoch 2/20
6704/6704 [==============================] - 1s 102us/step - loss: 0.4390 - accuracy: 0.8031 - val_loss: 0.3590 - val_accuracy: 0.8264
Epoch 3/20
6704/6704 [==============================] - 1s 96us/step - loss: 0.3802 - accuracy: 0.8364 - val_loss: 0.3021 - val_accuracy: 0.8820
Epoch 4/20
6704/6704 [==============================] - 1s 88us/step - loss: 0.3403 - accuracy: 0.8628 - val_loss: 0.2817 - val_accuracy: 0.8894
Epoch 5/20
6704/6704 [==============================] - 1s 99us/step - loss: 0.3130 - accuracy: 0.8747 - val_loss: 0.2649 - val_accuracy: 0.8970
Epoch 6/20
6704/6704 [==============================] - 1s 96us/step - loss: 0.3036 - accuracy: 0.8793 - val_loss: 0.2654 - val_accuracy: 0.9033
Epoch 7/20
6704/6704 [==============================] - 1s 96us/step - loss: 0.2930 - accuracy: 0.8847 - val_loss: 0.2545 - val_accuracy: 0.9036
Epoch 8/20
6704/6704 [==============================] - 1s 97us/step - loss: 0.2836 - accuracy: 0.8828 - val_loss: 0.2505 - val_accuracy: 0.9085
Epoch 9/20
6704/6704 [==============================] - 1s 96us/step - loss: 0.2786 - accuracy: 0.8911 - val_loss: 0.2488 - val_accuracy: 0.9068
Epoch 10/20
6704/6704 [==============================] - 1s 94us/step - loss: 0.2818 - accuracy: 0.8859 - val_loss: 0.2464 - val_accuracy: 0.9071
Epoch 11/20
6704/6704 [==============================] - 1s 95us/step - loss: 0.2800 - accuracy: 0.8880 - val_loss: 0.2488 - val_accuracy: 0.9099
Epoch 12/20
6704/6704 [==============================] - 1s 130us/step - loss: 0.2719 - accuracy: 0.8895 - val_loss: 0.2530 - val_accuracy: 0.9099
Epoch 13/20
6704/6704 [==============================] - 1s 102us/step - loss: 0.2705 - accuracy: 0.8926 - val_loss: 0.2498 - val_accuracy: 0.9071
Epoch 14/20
6704/6704 [==============================] - 1s 95us/step - loss: 0.2609 - accuracy: 0.8926 - val_loss: 0.2453 - val_accuracy: 0.9120
Epoch 15/20
6704/6704 [==============================] - 1s 93us/step - loss: 0.2644 - accuracy: 0.8923 - val_loss: 0.2430 - val_accuracy: 0.9113
Epoch 16/20
6704/6704 [==============================] - 1s 96us/step - loss: 0.2553 - accuracy: 0.8945 - val_loss: 0.2367 - val_accuracy: 0.9113
Epoch 17/20
6704/6704 [==============================] - 1s 97us/step - loss: 0.2614 - accuracy: 0.8913 - val_loss: 0.2448 - val_accuracy: 0.9081
Epoch 18/20
6704/6704 [==============================] - 1s 95us/step - loss: 0.2545 - accuracy: 0.8944 - val_loss: 0.2392 - val_accuracy: 0.9092
Epoch 19/20
6704/6704 [==============================] - 1s 94us/step - loss: 0.2514 - accuracy: 0.8939 - val_loss: 0.2464 - val_accuracy: 0.9085
Epoch 20/20
6704/6704 [==============================] - 1s 95us/step - loss: 0.2517 - accuracy: 0.9010 - val_loss: 0.2326 - val_accuracy: 0.9120
Train: 0.918, Test: 0.912
```





Accuracy : Train: 0.918, Test: 0.912

```
In [ ]:
```